Improved YOLOv3 on Edge Device for Object detection & Classification

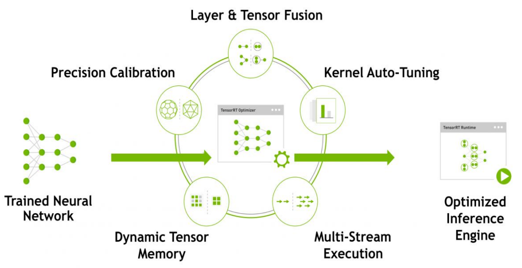
Siyu Liu, Elias Stein, John Sun {siyuliu3, elistein, js44}@stanford.edu

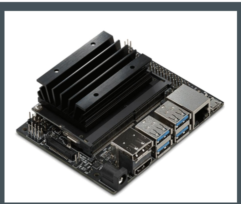
Project Summary

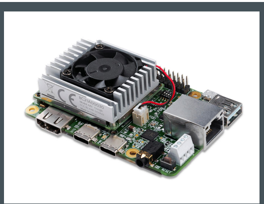
We explored post training optimization techniques for improving the development performance of object detection architectures such as YOLOv3 on resource constrained devices such as the Jetson Nano. Through the TensorRT framework we applied optimization techniques such as Loop Fusion, Kernel Tuning, and Quantization and measured the impact on performance metrics like accuracy and inference execution time.

These techniques all function around trying to reduce the memory and computational overhead of the model. For example, quantization does so by reducing the precision of the variables used in inference allowing for both faster computations (given the correct HW) and reduced memory.

We used the TensorRT framework from NVIDIA to apply these optimization techniques to YOLOv3 models of different sizes, trained on different datasets and deployed on different types of hardware.







Results

YOLOv3 - COCO - RTX 2060	No TensorRT (Baseline)		Loop Fusion + Kernel Tuning Optimizations				LF + KT + FP16 Quantization			
Size	Time / Inf (ms)	mAP	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change
320	27	52.48%	10	2.70x	35.35%	-32.64%	6	4.50x	35.37%	-32.60%
416	33	54.59%	16	2.06x	38.37%	-29.71%	8	4.13x	38.37%	-29.71%
608	64	55.16%	26	2.46x	38.60%	-30.02%	11	5.82x	38.60%	-30.0%

YOLOv3 - COCO - Jetson Nano	No TensorRT (Baseline)		Loop Fusion + Kernel Tuning Optimizations				LF + KT + FP16 Quantization			
Size	Time / Inf (ms)	mAP	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change
320	323	52.48%	249	1.30x	35.36%	-32.62%	154	2.10x	35.36%	-32.62%
416	500	54.59%	392	1.28x	38.24%	-29.95%	248	2.02x	38.24%	-29.95%
608	909	55.16%	805	1.13x	38.60%	-30.02%	501	1.81x	38.60%	-30.02%

YOLOv3 - VOC - RTX 2060	No TensorRT (Baseline)		Loop Fusion + Kernel Tuning Optimizations				LF + KT + FP16 Quantization			
Size	Time / Inf (ms)	mAP	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change	Time / Inf (ms)	Speedup Multiplier	mAP	mAP Change
320	26	75.10%	13	2.00x	64.33%	-14.34%	5	5.20x	64.32%	-14.35%
416	34	77.40%	20	1.70x	68.43%	-11.59%	7	4.86x	68.44%	-11.58%
608	62	76.64%	35	1.77x	68.92%	-10.07%	12	5.20x	68.95%	-10.03%

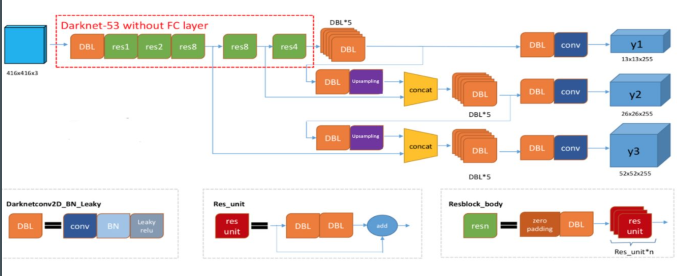
Future Work

- Repeat optimizations with a different framework to validate the accuracy drop measured by TensorRT.
- Utilize more aggressive INT8 quantization to examine more further examine trade-off of precision vs. performance.

References

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” inProceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp.779–788

“Deep Learning SDK Documentation,” *NVIDIA Tensorrt Developer Guide*, 06-Dec-2019. [Online].



Discussion

From the central figures, we see that our baseline measurements of YOLOv3 are as expected, roughly reproducing the mAP results reported by the original YOLOv3 paper. On both the RTX and Nano we pay significant speed costs, approximately 2x and 3x respectively, for marginal increases in mAP. The Nano fares worse given it's limited compute power and resulting sensitivity to increases in FLOPS.

First we measured YOLOv3 performance with for non-quantization optimizations like layer fusion (LF) and kernel tuning (KT). The speed increased significantly (2-3x for RTX and 1.2x for Nano) but the mAP score dropped significantly. This was quite surprising, since the applied optimizations should only work to increase arithmetic intensity and reduce loads and stores to memory. Interestingly, a review of NVIDIA discussion boards indicated that this may be due to a bug in TensorRT's implementation.

Next, we measure performance including 16-bit floating point quantization. We observed a negligibly small decrease in mAP with another significant boost to inference speed (~2.5x for RTX and Nano). This result is reasonable as we would expect that the error introduced by precision loss to be minimal in a large NN like YOLOv3 since less importance will be given to any one weight value. Furthermore, reducing precision loss will have a greater impact on small weight values (due to the nature of floating point), but these values inherently contribute less to the output calculation.

Overall, we found that applying optimization techniques such as quantization could provide for a 2x - 6x increase in inference execution speed. Such performance gains could be critical for real world deployment on an edge device. Of course, in our testing this came with a significant drop in accuracy, which may ultimately outweigh such speed gains